**Project Μαθήματος Ανάπτυξης Λογισμικού Για Πληροφορικακά Συστήματα**

**Αμπατζίδης Γεώργιος – 1115201400008**
**Βλάχος Νικόλαος – 1115201400023**
**Παναγιωτίδης Παναγιώτης – 1115201500200**

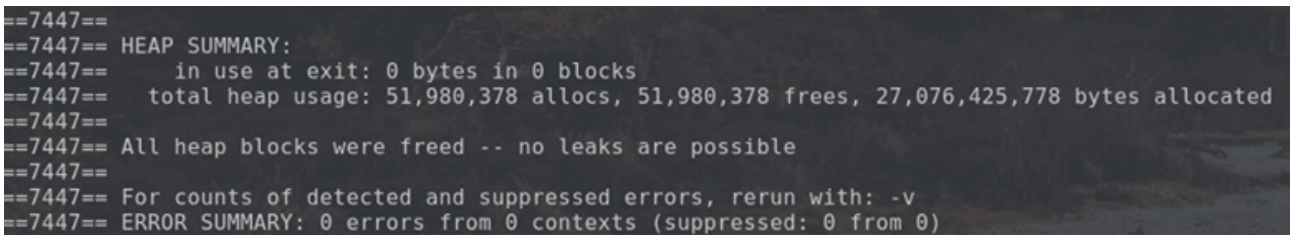
Με την έναρξη του προγράμματος μετά το φόρτωμα των relations κάνουμε initialize το thread pool, το οποίο παίρνει σαν όρισμα τον αριθμό των threads ο οποίος βρίσκεται ορισμένος στο include.h. Αρχικά δημιουργεί τη δομή thread pool που υπάρχουν μέσα της δομές και δημιουργούμε τόσα threads. Στο initialize του thread pool ορίζουμε μια μεταβλητή threads_alive ισο με 1. Στη συνέχεια περιμένουμε να ξεκινήσουν όλα τα threads πριν επιστρέψουμε στην main και με τη δημιουργία του κάθε thread πηγαίνει σε μια συνάρτηση, τη threadWork. Όταν το thread μπει στη threadWork κλειδώνει το mutex για πρόσβαση στο thread pool και αυξάνει τον μετρητή των live threads κατα 1. Όσο το πλήθος των εργασιών που βρίσκεται στο thread pool ειναι 0 το thread περιμένει μεχρι να δεχτεί σήμα(signal) στη μεταβλητή hasJobs του thread pool.Όταν έρθει και μπεί ενα job στο jobQueue στέλνει ενα σήμα(signal) σε όσες συναρτήσεις περιμένουν στη μεταβλητή hasJobs.

Τα threads κάνουν 3 βασικές δουλειές. Το histogram job, το partition job και το join job. Το histogram job παίρνει ενα chunk απο τον αρχικό πίνακα και δημιουργεί ενα ιστόγραμμα βασισμένο σε αυτό το κομμάτι.Το partition job παίρνει ενα κομμάτι του πίνακα μοιρασμένο ισόποσα σε ολα τα threads και κάνει reorder στον πίνακα με βάση τα τελευταία n ψηφία του payload. Για κάθε thread εχει δημιουργηθεί ενας pre fix sum πίνακας και κάθε thread καταλαμβάνει συγκεκριμένες θέσεις στον reordered πίνακα. Για παράδειγμα, έστω οτι το thread 1 έχει 5 στοιχεία που έχουν αποτέλεσμα hashing ίσο με το 0. Αυτά θα μπουν στις πρώτες 5 θέσεις. Μετά απο αυτά τα στοιχεία θα μπούν τα επόμενα n στοιχεία του επόμενου thread κοκ. Να τονιστεί πως δεν υπάρχει κάποιο data race για τις εγγραφές στον reordered πίνακα, καθώς οι θέσεις εγγραφών για το κάθε hash value απο το ασντίστοιχο thread ειναι προκαθορισμένες.Το join job παίρνει τα buckets απο τον index και τον non index πίνακα, βρίσκει ποιά payloads είναι ίσα μεταξύ τους και βρίσκει τα rowID. Για κάθε απο αυτά τα jobs υπάρχουν τα αντίστοιχα arguments που περνάνε στα threads τα στοιχεία που χρειάζονται για τους υπολογισμούς.

Το κάθε thread pool όταν γίνεται initialize κάνει initialize και ένα jobQueue, το jobQueue είναι μια διπλά συνδεδεμένη λίστα που περιέχει τα jobs που μπαίνουν απο τα διάφορα κομμάτια του προγράμματος. Έχει υλοποιηθεί επίσης η threadWait η οποία ελέγχει αν όλα τα threads έχουν σταματήσει όλες τις διεργασίες τους με το conditiond variable all idle.

Στην αρχή εκτέλεσης κάθε query φιλτράρονται τα κατάλληλα relations. Έπειτα, καλείται ο Query Optimizer, που αποφασίζει την καλύτερη αναμενόμενη σειρά εκτέλεσης των Join Predicates και με βαση τα αποτελέσματα που επιστρέφει εκτελούνται τα Joins. Ο Ouery Optimizer τον ακολουθεί τον αλγόριθμο του Join Enumeration, αποφασίζοντας τη σειρά εισαγωγής των relation στο koin tree με στόχο την ελαχιστοποίση των αναμενόμενων ενδιάμεσων αποτελεσμάτων. Το πλήθος των αναμενόμενων αποτελεσμάτων προβλέπεται με τη χρήση στατιστικών.
Με το πέρας των queries το οποίο σηματοδοτείται με newline στέλνουμε NULL functions στα threads σαν Job και θέτουμε το threads_alive = 0 Αυτό τερματίζει τα threads και κάνουνε join με το κύριο thread το οποίο αφού απελευθερώσει όποια μνήμη έχει δεσμεύσει τερματίζει.

Ο χρόνος υπολογισμού των στατιστικών ειναι μικρότερος απο 1 δευτερόλεπτο, όπως απαιτήται απο τις προδιαγραφές της άσκησης.




Χρόνοι εκτέλεσης πογράμματος με μεταβαλόμενο αριθμό thread και N όπου 2^N ο αριθμός των bucket.

| Num_Threads | N | Time (ms) |
|---|---|---|
| 8 | 5 | 4606 |
| 8 | 6 | 4543 |
| 8 | 7 | 4340 |
| 8 | 8 | 4512 |
| 9 | 5 | 4628 |
| 9 | 6 | 4621 |
| 9 | 7 | 4564 |
| 9 | 8 | 4451 |